SystemC Tutorial:
From Language to Applications,
From Tools to Methodologies

Grant Martin
Fellow, Cadence Berkeley Labs

SBCCI 2003, São Paolo, Brazil, 8-11 Sept 2003
8 September 2003: 0830-1030
Abstract

• This tutorial will cover SystemC from more than just a language perspective. It will start with a brief survey of language features and capabilities, including some of the more recent developments such as the SystemC Verification Library. The usage of several of these language features, in particular for system-level modelling, design, verification and refinement will be illustrated. We will then address many interesting applications of SystemC drawn from a number of different industrial and academic research groups.

• Next, we will talk about current tools available for design modelling, analysis and implementation with SystemC, covering the areas of cosimulation, synthesis, analysis, refinement, and testbenches, illustrating them with examples. Of course, tools are not enough; we will cover a number of methodology examples, in particular illustrating the use of SystemC in building complete design flows for complex SoC and system designs. This will also illustrate the linkage between SystemC and other design languages. We will close with a few notes on possible future SystemC evolution.
Outline

• The Context for SystemC
• Language Structure and Features
• Use Models
• Application Examples
• Tools
• Design Flows and Methodologies
• SystemC Futures
The Context for SystemC

(Hugo De Man’s “7th. Heaven of Software”)

System and SW Modeling: UML, SDL, etc.

System Level Integration Infrastructure: SystemC

Mere Implementation!! VHDL, Verilog, SystemVerilog

(Hugo De Man’s “Deep Submicron Hell of Physics”)
SystemC needs a ceiling as well as a floor

System and SW Modeling:
UML, SDL, etc.

Mere Implementation!!
VHDL, Verilog, SystemVerilog
How the Industry Looks at the Many Language Choices

SW and System Modelling

- **Best**
- **Good**
- **OK**
- **No**
- **No**
- **No**

Embedded SW Simulation

- **Best**
- **Good**
- **OK**
- **No**
- **No**

System Design

- **Best**
- **Good**
- **OK**
- **Best**
- **OK**
- **Best**

Verification

- **Best**
- **Good**
- **OK**
- **No**
- **No**

RTL

- **Best**
- **Good**
- **OK**
- **No**
- **No**

A Single Language Alone Cannot Effectively Cover All of the Design Flow

- **SystemC 2.01**
- **C/C++**
- **SystemVerilog**
- **SCVL, Vera, e**
- **VHDL/Verilog**
**SystemC is for System Level Design and Modeling**

- Real potential for SystemC is to be the industry standard language for system level design, verification and IP delivery for both HW and SW.

- Towards this goal, SystemC 2.0 supports generalized modeling for communication and synchronization with *channels, interfaces*, and *events*. Hardware signals are modeled as a specialization of channels.

- System level extensions in SystemC 2.0 support transaction-level modeling, communication refinement, executable specification modeling, HW/SW co-design.

- Ability to refine HW portions of design to RTL level within a single language is a unique strength of SystemC, as is the fixed point modeling capability, and easy integration of existing C/C++ models.
What are Users Doing Today with SystemC?

• A few user groups have experimented with or are using SystemC for RTL modeling, but this is not where the real interest is.

• Many companies/design groups are in the process of replacing in-house C/C++ system level modeling environments with SystemC.

• Many companies view SystemC as both a modeling language and a modeling “backplane” (e.g. for ISS integration).

• A number of companies have completed TLM & TBV modeling efforts using SystemC 2.0 and are very excited & interested. Some of the results are starting to be made publicly available. Some companies have announced that they will provide system-level IP using SystemC and have made it available:
Outline

• The Context for SystemC
• *Language Structure and Features*
• Use Models
• Application Examples
• Tools
• Design Flows and Methodologies
• SystemC Futures
SystemC 2.0 Language Architecture

Upper layers are built cleanly on lower layers.

Lower layers can be used without upper layers.

**Standard Channels for Various MOCs**
- Kahn Process Networks
- Static Dataflow, etc.

**Add-On Libraries**
- Verification Standard Library
- Master/Slave Library etc.

**Elementary Channels**
- Signal, Clock, Mutex, Semaphore, Fifo, etc.

**Core Language**
- Modules
- Ports
- Processes
- Interfaces
- Channels
- Events

**Data Types**
- Logic Type (01XZ)
- Logic Vectors
- Bits and Bit Vectors
- Arbitrary Precision Integers
- Fixed Point Numbers
- C++ Built-In Types (int, char, double, etc.)
- C++ User-Defined Types

**C++ Language Standard**
# SystemC Language recent updates

<table>
<thead>
<tr>
<th>Future (SystemC 3.0)</th>
<th>SW modeling: SW tasks and schedulers – RTOS modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under Investigation</td>
<td>Analog/mixed-signal modeling extension</td>
</tr>
<tr>
<td>Standard Channels for Models of Computation</td>
<td>Verification Standard Library</td>
</tr>
<tr>
<td>▪ Kahn process networks</td>
<td>▪ Transaction monitoring and recording</td>
</tr>
<tr>
<td>▪ Static dataflow</td>
<td>▪ Randomization and constraints</td>
</tr>
<tr>
<td>▪ Etc.</td>
<td>▪ HDL connection</td>
</tr>
<tr>
<td></td>
<td>▪ Data introspection</td>
</tr>
<tr>
<td>Elementary Channels</td>
<td>Data-Types</td>
</tr>
<tr>
<td>Signal, timer, mutex, semaphore, FIFO, etc.</td>
<td>▪ 4-valued logic types (01zx)</td>
</tr>
<tr>
<td></td>
<td>▪ 4-valued logic vectors</td>
</tr>
<tr>
<td></td>
<td>▪ Bits and bit-vectors</td>
</tr>
<tr>
<td></td>
<td>▪ Arbitrary-precision integers</td>
</tr>
<tr>
<td></td>
<td>▪ Fixed-point numbers</td>
</tr>
<tr>
<td></td>
<td>▪ C++ user-defined types</td>
</tr>
<tr>
<td>Core Language</td>
<td>Event-driven Simulation Kernel</td>
</tr>
<tr>
<td>▪ Modules</td>
<td>C++ Language Standard</td>
</tr>
<tr>
<td>▪ Ports</td>
<td></td>
</tr>
<tr>
<td>▪ Processes</td>
<td></td>
</tr>
<tr>
<td>▪ Events</td>
<td></td>
</tr>
<tr>
<td>▪ Interfaces</td>
<td></td>
</tr>
<tr>
<td>▪ Channels</td>
<td></td>
</tr>
</tbody>
</table>
Models of Computation in SystemC 2.0

• A model of computation is broadly defined by:
  – Model of time (real, integer, untimed) and event ordering constraints (globally ordered, partially ordered, etc.)
  – Methods of communication between processes
  – Rules for process activation

• Flexible communication and synchronization capabilities in SystemC 2.0 enable a wide range of MOCs to be naturally modeled.
  – Examples: RTL, Process Networks, Static Dataflow, Transaction Level Models, Discrete Event
  – These operate within the underlying event-driven kernel, although MOC-specific optimisations are possible – e.g. for all statically-scheduled dataflow, substitute a new kernel.
  – The open nature of SystemC allows many possible optimisations
RTL Model of Computation in SystemC

• Models combinational logic and sequential logic triggered by clocks.
• Very similar to RTL modeling in Verilog & VHDL.
• Signals modeled using sc_signal<> , sc_signal_rv<> 
• Ports modeled using sc_in<> , sc_out<> , sc_inout<>
Kahn Process Network MOC in SystemC

• Very useful for high level system modeling

• Modules communicate via FIFOs (sc_fifo<T>) that suspend readers and writers as needed to reliably deliver data items.

• Easy to use and guaranteed to be deterministic

• Pure KPN has no concept of time

• With annotated time delays, becomes timed functional model or performance model.
Static Dataflow MOC in SystemC

• A proper subset of the KPN MOC

• Each module reads and writes a fixed number of data items each time it is activated. Sample delays modeled by writing data items into FIFOs before simulation starts.

• Simulators and implementation tools can determine static schedule for system at compile-time, enabling high performance simulation and implementation.

• Commonly used in DSP systems, especially along with SystemC’s fixed point types (sc_fixed<>), sc_fix).
Transaction-Level MOC in SystemC

- Communication & synchronization between modules modeled using function calls (rather than signals)
- Transactions have a start time, end time, and set of data attributes (e.g. `burst_read(uint addr, char* data, uint n)`)  
- Two-phase synchronization scheme typically used for overall system synchronization
- Much faster than RTL models (more later…)

Communication between modules is modeled using function calls that represent transactions. No signals are used.

```
Read: Addr: 0xFF12
Data: 0x0123
```
```
Read: Addr: 0xFF14
Data: 0xBEEF
```
Modeling Example - Interfaces

class write_if : public sc_interface
{
    public:
        virtual void write(char) = 0;
        virtual void reset() = 0;
};

class read_if : public sc_interface
{
    public:
        virtual void read(char &) = 0;
        virtual int num_available() = 0;
};
Modeling Example - Channel

class fifo : public sc_channel, public write_if, public read_if
{
  public:
    fifo() : num_elements(0), first(0) {}

    void write(char c) {
      if (num_elements == max_elements)
        wait(read_event);

      data[ (first + num_elements) % max_elements ] = c;
      ++ num_elements;
      write_event.notify();
    }

    void read(char& c) {
      if (num_elements == 0)
        wait(write_event);

      c = data[first];
      -- num_elements;
      first = (first + 1) % max_elements;
      read_event.notify();
    }

    void reset() { num_elements = first = 0; }

    int num_available() { return num_elements; }

  private:
    enum e { max_elements = 10 };  // just a constant
    char data[max_elements];
    int num_elements, first;
    sc_event write_event, read_event;
};
Modeling Example - Producer / Consumer

class producer : public sc_module
{
    public:
        sc_port<write_if> out; // the producer's output port
    SC_CTOR(producer) // the module constructor
    {
        SC_THREAD(main); // start the producer process
    }
    void main() // the producer process
    {
        char c;
        while (true) {
            ...
            out->write(c); // write c into the fifo
            if (...) {
                out->reset(); // reset the fifo
            }
        }
    }
};

class consumer : public sc_module
{
    public:
        sc_port<read_if> in; // the consumer's input port
    SC_CTOR(consumer) // the module constructor
    {
        SC_THREAD(main); // start the consumer process
    }
    void main() // the consumer process
    {
        char c;
        while (true) {
            in->read(c); // read c from the fifo
            if (in->num_available() > 5) {
                ...; // perhaps speed up processing
            }
        }
    }
};
class top : sc_module
{
    public:
    fifo fifo_inst;          // a fifo instance
    producer *producer_inst; // a producer instance
    consumer *consumer_inst; // a consumer instance

    SC_CTOR(top)             // the module constructor
    {
        producer_inst = new producer("Producer1");
        // bind the fifo to the producer's output port
        producer_inst->out(fifo_inst);

        consumer_inst = new consumer("Consumer1");
        // bind the fifo to the consumer's input port
        consumer_inst->in(fifo_inst);  
    }
};
Communication Refinement in SystemC

- Channels may have multiple separate interfaces.
- Ports are bound to a particular interface, not to a channel.
- Interfaces can be reused with different channels.
- Communication can be refined via channel substitution.
- Examples of communication refinement:
  - Exploration during functional specification.
  - Retargeting abstract communication and synchronization to RTOS API.
  - Refining communication to a hardware implementation using adapters and hierarchical channels, perhaps followed by “protocol inlining”.
Transaction-Level Producer/Consumer Design

• Let’s start with an example design similar to the previous design:

```
<table>
<thead>
<tr>
<th>top</th>
</tr>
</thead>
<tbody>
<tr>
<td>producer</td>
</tr>
</tbody>
</table>
```
### Transaction-Level Producer/Consumer Design

```cpp
class producer : public sc_module
{
  public:
    sc_port<sc_fifo_out_if<char> > out;
    SC_HAS_PROCESS(producer);

    producer(sc_module_name name) :
      sc_module(name) {
        SC_THREAD(main);
    }

    void main() {
      const char *str =
        "Visit www.systemc.org!\n";
      const char *p = str;

      while (true) {
        if (rand() & 1) {
          out->write(*p++);
          if (!*p) p = str;
        }

        wait(1, SC_NS);
      }
    }
};

class consumer : public sc_module
{
  public:
    sc_port<sc_fifo_in_if<char> > in;
    SC_HAS_PROCESS(consumer);

    consumer(sc_module_name name) :
      sc_module(name) {
        SC_THREAD(main);
    }

    void main() {
      char c;

      while (true) {
        if (rand() & 1) {
          in->read(c);
          cout << c;
        }

        wait(1, SC_NS);
      }
    }
};
```
Transaction-Level Producer/Consumer Design

class top : public sc_module
{
public:
    sc_fifo<char> fifo_inst;
    producer prod_inst;
    consumer cons_inst;

    top(sc_module_name name, int size) :
        sc_module(name),
        fifo_inst("Fifo1", size),
        prod_inst("Producer1"),
        cons_inst("Consumer1")
    {
        prod_inst.out(fifo_inst);
        cons_inst.in(fifo_inst);
    }
};

int sc_main (int argc, char *argv[])
{
    int size = 10;

    top top1("Top1", size);
    sc_start(1000, SC_NS);
    cout << endl << endl;
    return 0;
}
• Assume we have the following RTL clocked HW FIFO model that we wish to insert into the just shown transaction-level producer/consumer design:
template <class T> class hw_fifo : public sc_module
{
  public:
    sc_in<bool>  clk;
    sc_in<T>     data_in;
    sc_in<bool>  valid_in;
    sc_out<bool> ready_out;

    sc_out<T>    data_out;
    sc_out<bool> valid_out;
    sc_in<bool>  ready_in;

  SC_HAS_PROCESS(hw_fifo);

  hw_fifo(sc_module_name name, unsigned size)
  : sc_module(name), _size(size)
  {
    assert(size > 0);
    _first = _items = 0;
    _data = new T[_size];

    SC_METHOD(main);
    sensitive << clk.pos();

    ready_out.initialize(true);
    valid_out.initialize(false);
  }

  ~hw_fifo() { delete[] _data; }

protected:

  void main()
  {
    if (valid_in.read() && ready_out.read())
    {
      // store new data item into fifo
      _data[(_first + _items) % _size] = data_in;
      ++_items;
    }

    if (ready_in.read() && valid_out.read())
    {
      // discard data item that was just
      // read from fifo
      --_items;
      _first = (_first + 1) % _size;
    }

    // update all output signals
    ready_out = (_items < _size);
    valid_out = (_items > 0);
    data_out = _data[_first];
  }

  unsigned _size;
  unsigned _first;
  unsigned _items;
  T* _data;
The hw_fifo_wrapper Hierarchical Channel

- We need to wrap the RTL hw_fifo module in order to use it in the transaction-level producer/consumer design:
The hw_fifo_wrapper Hierarchical Channel

template template <class T>
class hw_fifo_wrapper
: public sc_module, public sc_fifo_in_if<T>,
 public sc_fifo_out_if<T>
{
 public:
 sc_in<bool> clk;
 protected:
 // embedded channels
 sc_signal<T> write_data;
 sc_signal<bool> write_valid;
 sc_signal<bool> write_ready;
 sc_signal<T> read_data;
 sc_signal<bool> read_valid;
 sc_signal<bool> read_ready;

 // embedded module
 hw_fifo<T> hw_fifo_;

 public:
 hw_fifo_wrapper(sc_module_name name,
 unsigned size)
 : sc_module(name), hw_fifo_("hw_fifo1", size)
 {
 hw_fifo_.clk(clk);

 hw_fifo_.data_in (write_data);
 hw_fifo_.valid_in (write_valid);
 hw_fifo_.ready_out(write_ready);
 hw_fifo_.data_out (read_data);
 hw_fifo_.valid_out(read_valid);
 hw_fifo_.ready_in (read_ready);
 }

 virtual void write(const T& data)
 {
 write_data = data;
 write_valid = true;

 do {
 wait(clk->posedge_event());
 } while (write_ready != true);

 write_valid = false;
 }

 virtual T read()
 {
 read_ready = true;

 do {
 wait(clk->posedge_event());
 } while (read_valid != true);

 read_ready = false;
 return read_data.read();
 }

 virtual void read(T& d) { d = read(); }

 NOTE: See web link for System Design with SystemC book to download the complete source code.
class top : public sc_module {
  public:
    hw_fifo_wrapper<char> fifo_inst; // changed
    producer prod_inst;
    consumer cons_inst;
    sc_clock clk;                    // added

  top(sc_module_name name, int size) :
    sc_module(name),
    fifo_inst("Fifo1", size),
    prod_inst("Producer1"),
    cons_inst("Consumer1"),
    clk("c1", 1, SC_NS)          // added
  {
    prod_inst.out(fifo_inst);
    cons_inst.in(fifo_inst);
    fifo_inst.clk(clk);            // added
  }
};

• We can now simulate the RTL hw_fifo module within the transaction-level producer/consumer design!
  – The hw_fifo_wrapper read/write methods hide the detailed RTL hw_fifo signal protocol.
• The hw_fifo_wrapper read/write methods are closely related to transactors
Transaction-Level Modeling in SystemC

Communication between modules is modeled using function calls that represent transactions. No signals are used.

- Why do transaction-level modeling in SystemC?
  - Models are relatively easy to develop and use
  - HW and SW components of a system can be accurately modeled. Typically bus is cycle-accurate, and bus masters / slaves may or may not be cycle-accurate.
  - Extensive system design exploration and verification can be done early in the design process, before it’s too late to make changes
  - Models are fast – typically about 100K clock cycles per second, making it possible to execute significant amounts of the system’s software very early in the design process

- Transaction-level modeling is extensively covered in the System Design with SystemC book and the code for the simple_bus design is provided
Suggested Modelling Abstraction Levels

(Source: “Transaction Level Modeling: Overview and Requirements for SystemC Methodology” and “Introduction to TLM” by Mark Burton (ARM), Frank Ghenassia (STMicroelectronics and Stuart Swan (Cadence), May 13, 2003; and “ARM System-Level Modelling” by Jon Connell, June 25, 2003).

Algorithmic Level (AL)  
Foundation: Function  
Function-calls  
Functional

Programmer’s View (PV)  
Foundation: Memory Map  
Bus generic  
Architectural

Programmer’s View + Timing (PVT)  
Foundation: Timed Protocol  
Bus architecture  
Timing approx.

Cycle Level (CC)  
Foundation: Clock Edge  
Word transfers  
Cycle-accurate

RT Level (RT)  
Foundation: Implementation  
Signal/Bit  
Cycle-accurate
Transaction-Based Verification in SystemC

- Why do transaction-based verification in SystemC?
  - Ability to have everything (except perhaps RTL HDL) in SystemC/C++ provides great benefits: easier to learn and understand, easier to debug, higher performance, easy to integrate C/C++ code & models, open source implementation, completely based on industry standards
  - Allows you to develop smart testbenches early in the design process (before developing detailed RTL) to find bugs and issues earlier. Enables testbench reuse throughout the design process.
  - Much more efficient use of verification development effort and verification compute resources

Transaction-Based Verification in SystemC is described in the SystemC Verification Standard Specification, and in the documentation and examples included with the OSCI SCV reference implementation kit.
Outline

- The Context for SystemC
- Language Structure and Features
  - SystemC Verification Library
- Use Models
- Application Examples
- Tools
- Design Flows and Methodologies
- SystemC Futures
SystemC Verification Library (SCV) Standardisation

• Late 2001 - Early 2002:
  – Discussion on White Papers from Various Members
  – Requirement gathering, discussions, and prioritization

• April 2002 - August 2002
  – Creation of first proposal draft
  – Distribution of prototype codes and use scenarios
  – Discussion and revision on the proposal

• August 2002
  – Verification Working Group approved the SystemC Verification (SCV) standard specification version 1.0a

• September 2002
  – Steering Committee approved the SCV specification version 1.0a
Standardisation Activities, continued

• The SCV Reference Implementation
  – Cadence's TestBuilder team created a reference implementation, and used it to get feedback – layered on top of Core Language

• October 2002
  – OSCI LWG and VWG reviewing reference implementation

• Nov. 20, 2002: “Open SystemC Initiative Delivers SystemC Verification Library” (1.0, Beta – reference implementation made available OSCI web site)

• June 2003
  – SCV 1.0 Beta3 released

• Production likely for SCV 1.0 by September-October
Motivating Example

Overview of SCV Features

- Weighted randomization
- Constrained randomization
- Transaction recording
- Transaction recording
- HDL connection

**Data introspection:** Manipulation of data objects with arbitrary type (C/C++/SystemC types, user-specified composite types, enumeration types)
SCV provides APIs for creating Verification IP

Verification IP is designed for detecting bugs (e.g. a transactor for a AMBA bus)
* Consistent exception reporting mechanism
* Consistent debugging mechanism
Example: Data Introspection in SCV standard

```c
struct bus_data_t {
    unsigned addr;
    unsigned data;
};

// sharing a data object among multiple C++ threads
typedef scv_shared_ptr<bus_data_t> bus_data_h;

// importing a user-defined type into the SCV library
template<> scv_extensions<bus_data_t> : ... { ... }

// enabling PLI-like access to a data object with smart pointer to allow abstract operations (e.g. read/write values, traverse data structures or set callbacks on value changes)
typedef scv_smart_ptr<bus_data_t> bus_data_hh;
```
Example: Data Introspection for abstract operations

```cpp
scv_smart_ptr<int> k;  k->next();  //assigns the next random value to k.
```

Type access: (basis for attribute recording in transactions)

```cpp
unsigned scv_extensions_if :: get_num_fields() const; ...
```

Value access and assignment: (basis for attribute recording)

```cpp
void scv_extensions_if :: assign (long long);
long long scv_extensions_if :: get_integer() const; ...
```

Randomization: (basis for constrained randomization)

```cpp
void scv_extensions_if :: next(); ...
```

Callbacks: (basis for variable recording)

```cpp
void scv_extensions_if :: register_cb (...); ...
```
Transaction Recording

• Debugging at the transaction-level can speed up debugging and analysis time
• Each high-level operation indicated by the test represents a transaction
• A stream represents a set of related and overlapping transactions, typically w.r.t. the same interface.
• A generator represents a specific type of transactions within a stream.
• A transaction has begin-time, end-time, and attributes.
• A relation can be specified between two transactions.

\[\text{Read : ...} \quad \text{Write : ...} \quad \text{Read : ...} \quad \text{Proc\_Interrupt : ...}\]
Example: Transaction Recording in a Transactor

class master : public sc_module {

    scv_tr_stream  transaction_stream;
    scv_tr_generator<unsigned, unsigned> read_generator;

    unsigned do_read (unsigned addr) {
        bus_access_semaphore.wait(); wait(clk->posedge_event());
        scv_tr_handle h = read_generator.begin_transaction (addr);
        ...
        unsigned data = bus_data; wait(clk->posedge_event());
        read_generator.end_transaction (h , data);
        return data;
    }

};
Example: Simple Randomization

```c
void test_body() {
    scv_smart_ptr<bus_data_t> arg;
    arg->addr. keep_only ( 0x1000, 0xABCD); // restricts the range of values to be generated
    arg->data. keep_only ( 0,10 );
    for (int k=0; k<100; ++k) {
        arg -> next ( ); // generates a new random value
        master_p-> do_write(arg);
    }
}
```
Example: Creating a Simple Distribution

```cpp
scv_smart_ptr<int> p;
p->keep_only(0, 100);
p->keep_out(3, 98);
p->next();
```
Example : Creating a Complex Distribution

• Weighted randomisation : pick a value from a distribution specification

```cpp
scv_smart_ptr<int> p;
scv_bag<int> dist;
dist.add(0,16);
dist.add(1,8);
dist.add(2,4);
dist.add(3,2);
dist.add(4,1);
p->set_mode(dist);
p->next();
```
Example: Creating a Constraint

class write_constraint : virtual public scv_constraint_base {
public:

  scv_smart_ptr< bus_data_h > write;

  SCV_CONSTRAINT_CTOR(write_constraint) {
    SCV_CONSTRAINT( write->addr() < 0x00ff ); // write address is less than 255
    SCV_CONSTRAINT( write->addr() != write->data() ); // write address does not equal the data being written
    SCV_CONSTRAINT ( a() > b() && b() > c() && (a() – c() > 100) ); // complex constraint expression (of a,b,c)
  }
};

... 
write_constraint c("c"); c . next (); *p = *c.write; // style 1
p->use_constraint (c.write); p->next(); // style 2
SCV Constrained Randomisation

• Constrained randomisation: pick a value that satisfies the Boolean constraint or sets of constraints.

• A good use example is for ATM or IP packets: to ensure no packets point back to the sender, or there are none or a controlled number of invalid addresses, or to ensure an unbalanced traffic distribution to specific addresses.

• Characteristics of the SCV Constrained Randomisation Solver:
  – Distributes solutions uniformly over legal values
  – Good performance as number of variables grows
  – Commutability (order independence) of constraint equations
  – Can express complex constraints
  – Debugging of over-constrained (unsolvable) systems
  – Control value generation of constrained objects

• Reference: John Rose and Stuart Swan, “SCV Randomisation”, 8 August 2003. URL: http://www.testbuilder.net/reports/scv_random_white_paper_7aug03.pdf
Example: Callbacks

• A callback is called every time a value is assigned

```cpp
scv_smart_ptr<int> data;
data->register_cb(my_value_change_callbacks);
wait(1,SC_NS); *data = 3;
wait(1,SC_NS); data->next( ); // assigns a random value to data
wait(1,SC_NS); *data = 4;
```

3 \(\rightarrow\) 0x104 \(\rightarrow\) 4
Simulation Database

- **Signal information (VCD)**
  - RTL level semantic

- **Variable information**
  - Value change callbacks

- **Transaction information**
  - Stream and Generator
  - Begin time, end time
  - Attributes
  - Transaction Relation

- SCV Reference Implementation provides a primitive ASCII database.
- More complex capabilities can be provided in proprietary databases.
Callback Connection to Any Database

• SCV includes a set of callback registration routines
  – a proprietary database can be connected to any SystemC simulation
  – similar to how tools connect to a Verilog simulator through PLI.

```c
void my_database_init() {
    scv_tr_db::register_class_cb(database_cbf);
    scv_tr_stream::register_class_cb(stream_cbf);
    scv_tr_generator_base::register_class_cb(generator_cbf);
    scv_tr_handle::register_class_cb(handle_cbf);
    scv_tr_handle::register_special_attribute_cb(attribute_cbf);
    scv_tr_handle::register_relation_cb(relation_cbf);
}
```
Miscellaneous Additional Features

• HDL connection: a standard way to connect SystemC signals to an HDL signal identified by a character string
  
  \texttt{scv\_connect(sc\_signal<T>& s, const char * hdl, ...)}

  (Everything else, for example simulation control, is provided by tool vendors in specific tools)

• Exception Handling – Standard Reporting Methods
  
  \texttt{scv\_report::set\_actions(SCV\_INFO, SCV\_DO\_NOTHING);}
  
  \texttt{SCV\_REPORT\_ERROR("bad data", "the data in master ... " );}

• Debugging: SCV library has some classes to allow state query while debugging
  
  \texttt{gdb) data\_show()}
Outline

• The Context for SystemC
• Language Structure and Features
  – SystemC Verification Library
• **Use Models**
• Application Examples
• Tools
• Design Flows and Methodologies
• SystemC Futures
Design Space Exploration in System-Level Design

System-Level Design Questions

Do the components within the design work properly together?

How can the design be globally optimized?

How can the system-level design engineer be confident that the results obtained from design exploration will hold true when the system is implemented?
Validation of Transaction-Level Models

Diagram:
- Ethernet Test
- Ethernet Switch Abstract
- Enet Master TVM
- Ethernet Switch Verilog
- Enet Slave TVM
- Transaction Database
- TxE Search
- Transaction Equivalent?
- Performance Equivalent?
Functional Verification of Hardware

Is the final version of the design error-free?

Has all of the functionality of the design been proven to work correctly?

How can the verification engineer be sure that an error found in the design is a logical error instead of a performance error?
Embedded Software Verification Method: Hardware Model Abstraction

Abstract Hardware Model in SystemC 2.0
Faster Simulation

Software

μP
ISS

AHB Bus

AHB/APB Bridge

Memory Cntrl

SDRAM/ROM/FLASH Model

Int. Cntrl

Timer1

GPIO

UART

USB

Software Debugging Environment
Implement Abstract Module in RTL / Legacy RTL Method: Top-Down / Bottom-up Design

LAN

WAN

SystemC – Abstract Domain

Verilog – Signal Domain
Outline

• The Context for SystemC
• Language Structure and Features
  – SystemC Verification Library
• Use Models
• "Application Examples"
• Tools
• Design Flows and Methodologies
• SystemC Futures
Application Examples – Useful Reference

A Method for the Development of Combined Floating- and Fixed-Point SystemC Models

Yves Vanderperren
yves.vanderperren@mie.alcatel.be

5. European SystemC Users Group Meeting

ALCATEL microelectronics

05/03/2002

All rights reserved
© 2002 - Alcatel Microelectronics
\[ \rho_{xx}(n) = \sum_{k=0}^{L-1} x(n-L-k) \cdot x^*(n-k) \]
Experiences and Challenges of Transaction-Level Modelling with SystemC 2.0

Alain CLOUARD
STMicroelectronics
Central R&D – Crolles (Grenoble, France)
MPEG4 SoC Transactional Model

Objective
Enable application software development *concurrently* with HW design

Users

Used by MPEG4 IVT team for software development 6 months before RTL top netlist ready

Used by SoC team for ARM software development

[Image of a diagram showing the transactional model]
MPEG4 SoC Transactional Model

Key benefits
- Close to emulation speed
- Debug facilities
- Fast development (days to few mm)

Simulation time for 1 image (coding + decoding)
A Design Methodology for the Development of a Complex SoC using UML and Executable System Models

Yves Vanderperren
yves.vanderperren@st.com

6th European SystemC User Group Meeting
October 22nd, 2002
From Function to Architecture

System functionality can be modelled in abstract terms using conventional OOA approach.

UML can be tailored to represent SystemC abstractions such as Module, Port, Channel (e.g. using stereotypes).

A Design Methodology for the Development of a Complex SoC using UML and Executable System Models, © 2002 - ST Microelectronics
SystemC – AMS Study Group

SystemC – Analog and Mixed Signal

Karsten Einwich
SystemC-AMS concept

User view layer
  View 1.1  View 1.2  ...  View 2.1  ...

Solver layer
  Solver 1  Solver 2  ...

Synchronization layer
  AMS - Synchronization

SystemC layer
  SystemC kernel

classical SystemC Layers
[1]
Networked embedded devices design:
NS-SystemC timing-accurate synchronization
Simulation cores synchronization

Network Simulator by Berkeley

Programming Model

Time synchronization

Data Exchange

Performance
## Data Exchange

<table>
<thead>
<tr>
<th>Packet Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Type</td>
</tr>
<tr>
<td>Data time</td>
</tr>
<tr>
<td>Data Size</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Receiver</td>
</tr>
</tbody>
</table>

... Next Event Time

[Diagram showing process queue wrapper, NS2 kernel, data queue manager, SystemC kernel, process queue wrapper, and message flow]

All rights reserved

European SystemC Users Group Meeting

7
Extending the SystemC™ Synthesis Subset by Object Oriented Features
Experiences and Results from the ODEtte Project

Eike Grimpe
OFFIS Research Institute

SystemC Technological Symposium,
DAC’03
Why?

- Today:
  SystemC synthesis subset \(\equiv\) HDL synthesis subsets
  - why should anyone use it for synthesis at this level?
- Extending the synthesis subset by OO features seems to be a logical step
- high level spec. \(\downarrow\)
synthesisable spec.
  \{ time consuming
  - error-prone
- Without improved synthesis techniques, SystemC will remain ‘only’ a design space exploration language
**SystemC™**

+ modules
+ signals & ports
+ processes
+ data types
  - hier. channels
  - sc_fifo, sc_mutex, ...
  - dynamic threads

- eliminate inheritance
- replace objects
- replace member access
- resolve polymorphism
- instantiate templates
- no behavioral synthesis

- ‘flattened’ behavioral/RT level SystemC or VHDL
- processable with existing tools

**OOHWLIBLE**

+ polymorphism
+ shared objects & inter-process communication

**C++**

+ classes
+ templates
+ inheritance
  - pointers
  - file I/O
  - new, delete

**VHDL**

+ functional equivalent
+ cycle accurate

SystemC Technological Symposium, DAC'03
Enabling system analysis of TI c55x processor megacell based designs via integration with OCP SystemC testbench

Saurabh Tiwari (saurabh@ti.com)
Software Design Engineer
Texas Instruments India Ltd.
C55x TL Architecture Exploration
Outline

• The Context for SystemC
• Language Structure and Features
  – SystemC Verification Library
• Use Models
• Application Examples
• **Tools**
• Design Flows and Methodologies
• SystemC Futures
Taxonomy of “SystemC EDA products” from OSCI web pages

- Total number of products = 38 (last update 1 June 2003) (# was 32 – 9 August 2002)
  - Commercial SystemC Simulators: 4 (3)
  - Co-Simulators: 4 (4)
  - Links to Emulation: 3 (4)
  - Synthesis: 6 (6)
  - HDL to SystemC Model Converters: 3 (4)
  - SystemC Extended Libraries: 4 (2)
  - Analysis, Display, Verification and Checkers: 3 (3)
  - System Level Modelling and Design Tools: 10 (6)
Examples of Tools

• Commercial SystemC Simulators
  – Cadence, Forte, Synopsys, Veritools

• Co-Simulators
  – Cadence, Mentor, Synopsys, TNI-Valiosys, (Celoxica)

• Links to Emulation
  – Dynalith, EVE, Mentor (IKOS)

• Synthesis
  – Adelante (ARM), Forte, Prosilog, Synopsys, Xilinx

• HDL to SystemC Model Converters
  – Ascend, Tenison, TNI-Valiosys
Examples of Tools, continued

• SystemC Extended Libraries
  – Adelante (ARM), ARM, Forte, Simucad

• Analysis, Display, Verification and Checkers
  – Actis, Blue Pacific, Verisity, (ChipVision ORINOCO system level power estimation)

• System Level Modelling and Design Tools
  – Axys Design, Cadence, CoWare, Future Design Automation, LisaTek (CoWare), Prosiolog, Summit Design, Synopsys
Example of Tool – Cadence SPW 4.8

HDL – Verilog, VHDL

Control Entry
Signal Analysis
Data Path Entry
Block Wizard
Verilog AMS

Software on ISS
Integrated Debug
Cross Debug C/C++/RTL
HDL Waveform
Signal Analysis
Example of Tool: Cadence Incisive SystemC

ARM Functional Virtual Prototype SystemC Model in Cadence Incisive Browser
Example of Tool: User-programmed SystemC analysis “widgets”

ARM LCD and Memory Display Widgets Linked to Their SystemC Model
Example of Tool: User-programmed SystemC analysis “widgets”

ARM Memory Transaction Level Model
Interactive Debug Window
Example of Tool: Display in Cadence Incisive SystemC
Example of Tool: Transaction-Level analysis in Cadence Incisive SystemC

Cadence SimVision Display of Transaction Explorer
Example of Tool: Synopsys CoCentric System Studio (used in TI example earlier)

C55x TL Architecture Exploration
Example of Tool: CoWare ConvergenSC
System-level design and verification

Cache Hits/Misses and SW Task Gantt

Memory Reads and Writes

Transaction Counts and Bus Contention
Example of Tool: Forte Cynthesizer (SystemC Behavioural Synthesis)
Example of Tool: ChipVision ORINOCO System-level Power Estimation

ORINOCO® leads to your best decisions.

- Analyze and understand your own design
- ORINOCO provides extensive visualization features to give you an insight into the power impact of your design
- It depicts the impact of your design moments after the changes were made.

**Identification of Hot Spots**

- This example identifies the imcOBA as the real Hot Spot
- The Power Burner has the highest potential for optimization.

**Data Encoding**

- Using simple data encoding, this example shows 20% power savings in success.

**ORINOCO® BENEFITS**

<table>
<thead>
<tr>
<th>Benefit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI - and Symbolic design entry</td>
<td>Integration with leading industry design tools and simulators</td>
</tr>
<tr>
<td>Real, behavioral estimation</td>
<td>Generation of power models with minimum user intervention</td>
</tr>
<tr>
<td>Bound estimation for the architectural power design space</td>
<td>Example IP interface</td>
</tr>
</tbody>
</table>
| Extensive visualization capabilities | Linux, Solaris, and Red Hat
| Low power architecture advisor | Linux 2.6, Solaris 7.3 |
| High Performance for large designs | |

**Memory Mapping - Wavelet Transform (10)**

Optimization of memory accesses and mapping result in substantial power savings.

**Algorithm Transformation - DIFFEQ Benchmark**

Easy algorithm transformation presents very different power results.

- Power Gain: 33.1% Total: 20%
Future SystemC Tool Possibilities

• A *Personal* View:

  – Links to Implementation are important
    – But the world has not figured out behavioural synthesis yet (although a next
geneneration of behavioural synthesis, and coprocessor synthesis, is emerging)
    – And using SystemC as an RTL entry vehicle is not the best approach
  – System level modelling, analysis and refinement is still not a well-understood
  and well-adopted approach
    – This is where users of SystemC need to spend most of their time, experimenting and
  working out methodologies
  – Calls out for:
    – Methodology-driven design flows
    – Analysis capabilities
    – Design space exploration concepts
    – Flows from higher level modelling e.g. UML, and links to embedded SW
  – From the system level designer viewpoint, this is the most useful area for tool
development
Outline

• The Context for SystemC
• Language Structure and Features
  – SystemC Verification Library
• Use Models
• Application Examples
• Tools
  • Design Flows and Methodologies
• SystemC Futures
Design Flows with SystemC: 2 key decisions

• Where You Start
  – Some other high level modelling language or tool
    – E.g. UML, SDL, Matlab/Simulink
  – Functional model in SystemC
    – E.g. Untimed or Timed Function (UTF, TF)
  – Architectural
    – Functional or Transaction-level model of the system implementation architecture

• How You Go
  – Model-Refine-Synthesise
    – (to SystemC RTL, HDL RTL, or HDL Gates)
  – Model-Refine-Manually transfer
    – (to SystemC RTL or HDL RTL)

In addition, for Derivative Design/Embedded SW Design and Verification: Building a model upwards from a SystemC architectural or implementation model (Platform model)
Possible Flows

Higher level modelling: UML, SDL, Matlab/Simulink

Functional

Architectural

SystemC Code Generation

Refine: e.g. transaction-level

Synth  Man

RTL

Implement

Possible Entry Points

Implementation Routes

Possible Flows

SystemC, Verilog, VHDL, Verilog-2005, SystemVerilog

Verilog, VHDL, Verilog-2005, SystemVerilog

Platform Model For System/SW Verification
Flows starting with Higher-level languages or notations

• UML:
  – ST (Alcatel) UML flow shown earlier
  – UML Code Generation for SystemC: (Yves Vanderperren, 6th. European SystemC users group meeting)
    – “SoC Design with UML and SystemC”, Alberto Sardini, Rational, 6th. European SystemC users group meeting
    – “A SystemC based design flow starting from UML Models”, Bruschi, Politecnico di Milano, 6th European SystemC users group meeting

• Matlab/Simulink:
  – “Modeling Cycle-Accurate Hardware with Matlab/Simulink Using SystemC”, Czerner and Zellmann, Ilmenau, 6th. European SystemC users group meeting
A SystemC based design flow starting from UML models

Politecnico di Milano, Cefriel
Siemens ICM
Front-end system design flow

- Specification document
  - Functional verification
  - SystemC specification for synthesis
    - Design space exploration
      - SystemC hw spec
      - C++ sw spec
  - UML model (analysis and implementation)
    - CoCentric System Studio
Fujitsu – UML, SystemC

New SoC Design Methodology

Software Algorithms
C/C++

System Specifications

Requirements, Standards

UML/SystemC

Division of HWSW Work
Performance Evaluation

Optimized Architecture
by Quantitative Analysis

Hardware Spec.

Verification Environment for
Levels of Abstraction/Timing Accuracy

Software Spec.

HWSW Co-Verification

Final Code

RTOS: Real Time Operating System
Untimed: Behavioral Model without
Clock Definition
Timed: Behavioral Model with (partial)
Clock Definition

Un timed

Timed

RTL

Gate Level, P&R Implementation

Hardware Refinement

RTOS Mapping

Final Code

All Rights Reserved. Copyright © FUJITSU LIMITED
Complete SystemC-based flow

- Modelling in SystemC
- Refining in SystemC
- Verification in SystemC
- Manual Translation to Verilog (currently)
- Synthesis from Verilog (currently)
- Eventual goal: Synthesis from SystemC at RTL and (perhaps) transaction-level
- Rob Slater, Motorola Israel, “Towards a complete SystemC flow”, 6th. European SystemC users group meeting
Towards a Complete SystemC Flow

Rob Slater
Motorola Semiconductor Israel, Ltd. (MSIL)
r.slater@motorola.com
A SystemC Based Flow

SystemC (C++) Model

Refine

Analysis

Results

Refine by Hand

SystemC

Synthesis

Rest of Process

Verilog

☑ Done By Designers

☑ Netlist Preserved
SystemC for Verification
Example of Tool: Axys Design MaxSIM Developer Suite – System Platform Model Creation and Export
Outline

• The Context for SystemC
• Language Structure and Features
  – SystemC Verification Library
• Use Models
• Application Examples
• Tools
• Design Flows and Methodologies
• SystemC Futures
SystemC 2.1

- SystemC 2.1 intended as a relatively minor release to add features that could not wait until SystemC 3.0
  - Intended to have very high compatibility with SystemC 2.0.1
  - Specs and code for 2.1 were developed by LWG over last year (2002-2003)
  - Anticipated availability sometime 2H 2003 – perhaps October/November

- Main features
  - Dynamic Thread Creation (designed with SystemC 3.0 in mind) – also critical for testbenches (e.g. SCV) and general SW modelling
  - New error reporting API
  - Exported Interfaces
  - A variety of small cleanups and bug fixes
SystemC 3.0

- SystemC 3.0 will be a major release that adds RTOS and scheduler modeling capabilities such as:
  - Thread interrupt and abort
  - User-defined scheduler models layered on top of the core SystemC scheduler
- (as indicated, requires dynamic thread creation for SW/RTOS modelling)
- Status: 3.0 specification to be continued after 2.1 is finished and IEEE SystemC standardisation (based on 2.01) started – thus, likely to continue in 2004. Plans are not firm at this point.
### Layered Language Architecture

<table>
<thead>
<tr>
<th>libraries</th>
<th>RTOS models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>user-defined channels</td>
</tr>
<tr>
<td></td>
<td>Scheduler models</td>
</tr>
<tr>
<td>core language</td>
<td>elementary channels</td>
</tr>
<tr>
<td>kernel</td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td>channels &amp; interfaces</td>
</tr>
<tr>
<td></td>
<td>scheduler API</td>
</tr>
<tr>
<td></td>
<td>events &amp; sensitivity</td>
</tr>
<tr>
<td></td>
<td>dynamic threads &amp; thread control</td>
</tr>
<tr>
<td></td>
<td>kernel scheduler</td>
</tr>
</tbody>
</table>

**Source:** Thorsten Groetker, “Modelling software with SystemC 3.0”, 6th. European SystemC Users group meeting, October, 2002
Beyond SystemC 3.0 – tentative roadmap

• At one point there was the idea of SystemC 4.0 with Analogue/Mixed-Signal modelling and solver capabilities (cf. SystemC-AMS study group and earlier presentation by Karsten Einwich)
  – Status: might continue as community effort
• Donation of SystemC (based on 2.01 Language Reference Manual) from OSCI to IEEE for official standardisation – likely by late 2003/early-2004. (2.01 LRM on OSCI web)
• Other OSCI Working Groups
  – Transaction-Level Modelling – standardise semantics, and perhaps APIs, for agreed levels of transaction-level models. Preliminary standards possible Q1-2 2004.
    – Leveraging work with ARM AMBA, OCPIP, and other developments
  – Synthesis subset of SystemC- behavioural and RTL. Draft for review by Oct-Nov.
  – Verification library (SCV) may also be donated to IEEE for standardisation
• Future of OSCI:
  – May become a usage and idea development community
  – When SystemC standardised by the IEEE, OSCI may (or may not) withdraw from developing reference implementations (possible alternative: “community prototypes”)
    – May leave this for commercial tool vendors (cf. Verilog, VHDL)
Conclusions

• SystemC is very clear a system-level modelling language
• Can be used as the basis for system-level design, verification and implementation flows
• Not a substitute for HDLs
• Is being applied in real-life design situations and being used to build real system-level design tools, methodologies and flows
• Its open nature, and being based on C++, allows many variant applications and flows to be built
• Can be easily plugged into both higher and lower level modelling and implementation environments
• Is being extended by the community in several interesting directions
• Has a very interesting future!